HW3. Matrix Multiplication

2020147555 컴퓨터과학과 강종서

적용한 방식
1. Loop collapse & openMP
matmul.cpp 의 matmul_ref() 함수의 nested loop 를 풀어준 뒤 openMP를 활용해서 멀티쓰레드로 실행을 하였고, 실행 시간은 50초대가 나왔다.

2. B Transposed
1번 코드에서 B의 transpose를 구한 뒤 이를 계산에 활용해 주었다. Transpose를 구하는 과정 역시 loop collapse 해준 뒤 openMP를 활용해 멀티쓰레드로 실행해주었다. 실행 속도는 대략 15초 정도 향상된 결과를 보였다.

```
[mgp2023_20@workspace-pow48ro1b97w-0:~/HW3/matmul$ make remote
scp src/matmul.cpp mgp-run:/home/mgp2023_20/HW3/matmul/src
matmul.cpp
ssh mgp-run "cd HW3/matmul && make" | tee result/eval_output.txt
g++ -std=c++11 -pthread -lpthread -fopenmp -O3 -Wl,--no-as-needed -mar
g++ -std=c++11 -pthread -lpthread -fopenmp -O3 -Wl,--no-as-needed -mar
./matmul /HW3_data/input.dat /HW3_data/output.dat | tee result/eval_ou
====================================
         Matrix Multiplication
====================================
The size of Matrix: 2048
====================================

Read input file(/HW3_data/input.dat)...
Read output file(/HW3_data/output.dat)...

Run your solution...

matmul_optimal took 34.5934 sec
Correct
```

3. B Transposed & block with locks
2번 코드에서 각각의 matrix를 block으로 쪼개어 계산하도록 해주었다. 하지만 행렬을 block으로 쪼개는 과정에서 순서를 고려해 주지 않아서 서로 다른 쓰레드에서 C 배열의 동일한 위치를 참조해 잘못된 결과가 나왔다.C의 block을 단위로 걸어주었고 lock을 걸어주어 해당 오류를 해결했다.

```
for(long long xyz = s; xyz < e; xyz++){
        x = xyz / (N * M);
        y = (xyz / M) % N;
        z = xyz % M;
        for(int ii = 0; ii < n_div; ii++){
                for(int jj = 0; jj < n_div; jj++){
                        for(int kk = 0; kk < m_div; kk++){
                                i = x * n_div + ii;
                                j = y * n_div + jj;
                                k = z * m_div + kk;
                                if(i < n && j < n && k < m){
                                        lock_guard<mutex> guard(mutexes[x][y]);
                                        C[i * n + j] += A[i * m + k] * BT[j * m + k];
                                }
                        }
                }
        }
}
```

실행 속도는 2번에서 20초 정도 향상된 결과를 보였다.

```
[mgp2023_20@workspace-pow48ro1b97w-0:~/HW3/matmul$ make remote
scp src/matmul.cpp mgp-run:/home/mgp2023_20/HW3/matmul/src
matmul.cpp
ssh mgp-run "cd HW3/matmul && make" | tee result/eval_output.tx
g++ -std=c++11 -pthread -lpthread -fopenmp -O3 -Wl,--no-as-need
g++ -std=c++11 -pthread -lpthread -fopenmp -O3 -Wl,--no-as-need
./matmul /HW3_data/input.dat /HW3_data/output.dat | tee result/
====================================
         Matrix Multiplication
====================================
The size of Matrix: 2048
====================================

Read input file(/HW3_data/input.dat)...
Read output file(/HW3_data/output.dat)...

Run your solution...

matmul_optimal took 13.3879 sec
Correct
```

4. B Transposed & block with no locks
3번의 코드에서 for loop를 수정해 하나의 쓰레드에서 C의 block 하나를 온전히 계산하도록 변경해주었다. 서로 C의 같은 위치를 참조하는 일이 없어졌고 기존의 lock을 전부 제거해주었다. block의 사이즈는 하나의 block이 전부

L1 캐시에 저장될 수 있도록 정해주었다. 실제로 block의 크기가 200을 넘어가면 실행 속도가 느려짐을 확인할 수 있었다.

```
void block_mul(const int* const A, const int* const BT, int* const C,
                const int n, const int m, const long long N, const long long M,
                const int n_div, const int m_div, const long long x, const long long y)
{

        for(long long z = 0; z < M; z++){
                for(int ii = 0, i = x * n_div; ii < n_div && i < n; i++, ii++){
                        for(int jj = 0, j = y * n_div; jj < n_div && j < n; j++, jj++){
                                for(int kk = 0, k = z * m_div; kk < m_div && k < m; k++, kk++){
                                        C[i * n + j] += A[i * m + k] * BT[j * m + k];
                                }
                        }
                }
        }
}
```

실행 속도는 약 3초의 결과를 보였다.

```
mgp2023_20@workspace-pow48ro1b97w-0:~/HW3/matmul$ make remote
scp src/matmul.cpp mgp-run:/home/mgp2023_20/HW3/matmul/src
matmul.cpp
ssh mgp-run "cd HW3/matmul && make" | tee result/eval_output.tx
g++ -std=c++11 -pthread -lpthread -fopenmp -O3 -Wl,--no-as-need
g++ -std=c++11 -pthread -lpthread -fopenmp -O3 -Wl,--no-as-need
./matmul /HW3_data/input.dat /HW3_data/output.dat | tee result/
=====================================
         Matrix Multiplication
=====================================
The size of Matrix: 2048
=====================================

Read input file(/HW3_data/input.dat)...
Read output file(/HW3_data/output.dat)...

Run your solution...

matmul_optimal took 3.30794 sec
Correct
```

5. B Transposed & block & Loop unrolling with AVX2
4번의 코드에서 k의 값이 변경되는 for loop를 unrolling 하여 반복 횟수를 8분의 1로 줄였다. 그 다음 A와 B의 8개의 원소를 vector에 저장 한뒤 AVX2 의 _mm256_mullo_epi32() 함수를 사용해 곱해주었다.[1] 그리고 결과이 결과를 C에 누적해 더해주었다. 하지만 본인의 구현 미숙으로 인해 크게 속도가 향상 되지는 않았다. Local 컴퓨터로 실행하였을 때는 2초대 초반의 결과를 보였지만 server 컴퓨터로 실행 시 대략 5초의 결과를 보였고 때문에 본 과제에서는 4번 코드를 제출한다.

발생한 문제점과 해결 과정
1. openMP를 활용하기 위해 loop collapse 를 진행할 때 변수의 값이 int를 넘어 overflow가 발생했었다. 초기에는 원인을 찾지 못하고 고민 하다가 작은 matrix에서는 오류 없이 작동함을 확인해 overflow가 발생하는 변수들을 long long으로 변환해주었다.

2. C의 같은 block의 값을 구하는 식들이 여러 쓰레드에 나누어져 있어 제대로 된 결과가 나오지 않았다. 초기에는 openMP의 내부 처리 과정으로 인한 오류라고 생각해 pthread를 통해 구현하였으나 동일한 오류가 발생했다. 위와 같은 원인으로 인한 오류임을 파악한 뒤 lock을 걸어 문제를 해결했고, 추후에 for loop의 구조를 변경해 lock을 제거해 주었다.

참고 문헌
Matt Scarpino, “Crunching Numbers with AVX and AVX2”, codeproject, 2023-05-03, https://www.codeproject.com/Articles/874396/Crunching-Numbers-with-AVX-and-AVX

1 Matt Scarpino, “Crunching Numbers with AVX and AVX2”, codeproject, 2023-05-03, https://www.codeproject.com/Articles/874396/Crunching-Numbers-with-AVX-and-AVX